Vitesses d'exécution dans R : comparaisons de base R, dplyr et data.table

Antoine Sireyjol

14 février 2019

Plan de la présentation

Plan de la présentation

Plan de la présentation

Plan de la présentation

Plan de la présentation

Dplyr et data.table

Dplyr et le tidyverse

- Tidyverse : environnement d'analyse de données en R

Dplyr et le tidyverse

- Tidyverse : environnement d'analyse de données en R
- Propre format de données : le tibble

Dplyr et le tidyverse

- Tidyverse : environnement d'analyse de données en R
- Propre format de données : le tibble
- Syntaxe caractéristique et concurrente des fonctions de base R avec `dplyr`

Dplyr et le tidyverse

- Tidyverse : environnement d'analyse de données en R
- Propre format de données : le tibble
- Syntaxe caractéristique et concurrente des fonctions de base R avec `dplyr`
- Chaînage possible des instructions avec %>%

Dplyr et le tidyverse

- Tidyverse : environnement d'analyse de données en R
- Propre format de données : le tibble
- Syntaxe caractéristique et concurrente des fonctions de base R avec `dplyr`
- Chaînage possible des instructions avec %>%
- Très lisible et optimisé

Syntaxe dplyr (1)

La grammaire `dplyr` s'appuie sur des fonctions aux noms explicites :

- `mutate(data, newvar1 = fonction(var1, var2...))` et `transmute(data, newvar1 = fonction(var1, var2...))` créent de nouvelles variables

Syntaxe dplyr (1)

La grammaire `dplyr` s'appuie sur des fonctions aux noms explicites :

- `mutate(data, newvar1 = fonction(var1, var2...))` et `transmute(data, newvar1 = fonction(var1, var2...))` créent de nouvelles variables
- `filter(data, condition)` sélectionne au sein d'une table certaines observations.

Syntaxe dplyr (1)

La grammaire `dplyr` s'appuie sur des fonctions aux noms explicites :

- `mutate(data, newvar1 = fonction(var1, var2...))` et `transmute(data, newvar1 = fonction(var1, var2...))` créent de nouvelles variables
- `filter(data, condition)` sélectionne au sein d'une table certaines observations.
- `arrange(data, var1, descending var2,...)` trie une base selon une ou plusieurs variables.

Syntaxe dplyr (1)

La grammaire `dplyr` s'appuie sur des fonctions aux noms explicites :

- `mutate(data, newvar1 = fonction(var1, var2...))` et `transmute(data, newvar1 = fonction(var1, var2...))` créent de nouvelles variables
- `filter(data, condition)` sélectionne au sein d'une table certaines observations.
- `arrange(data, var1, descending var2,...)` trie une base selon une ou plusieurs variables.
- `select(data, var1 : varX)` sélectionne certaines variables dans une base.

Syntaxe dplyr (1)

La grammaire `dplyr` s'appuie sur des fonctions aux noms explicites :

- `mutate(data, newvar1 = fonction(var1, var2...))` et `transmute(data, newvar1 = fonction(var1, var2...))` créent de nouvelles variables
- `filter(data, condition)` sélectionne au sein d'une table certaines observations.
- `arrange(data, var1, descending var2,...)` trie une base selon une ou plusieurs variables.
- `select(data, var1 : varX)` sélectionne certaines variables dans une base.
- `group_by(data, var)` regroupe une table par une variable

Syntaxe dplyr (1)

La grammaire `dplyr` s'appuie sur des fonctions aux noms explicites :

- `mutate(data, newvar1 = fonction(var1, var2...))` et `transmute(data, newvar1 = fonction(var1, var2...))` créent de nouvelles variables
- `filter(data, condition)` sélectionne au sein d'une table certaines observations.
- `arrange(data, var1, descending var2,...)` trie une base selon une ou plusieurs variables.
- `select(data, var1 : varX)` sélectionne certaines variables dans une base.
- `group_by(data, var)` regroupe une table par une variable
- `summarise(data, newvar1 = mean(var1), newvar2 = sum(var2))` réalise toute sorte d'opérations statistiques sur une table.

Syntaxe dplyr (2)

- Possibilité de chaîner ces opérations : l'opérateur %>%

Syntaxe dplyr (2)

- Possibilité de chaîner ces opérations : l'opérateur %>%
- `fonction(data, params...)` est équivalent à `data %>% fonction(params...)`

Syntaxe dplyr (2)

- Possibilité de chaîner ces opérations : l'opérateur %>%
- `fonction(data, params...)` est équivalent à `data %>% fonction(params...)`
- Exemple :

Syntaxe dplyr (2)

```
library(tidyverse)
# on crée un data frame avec 100 lignes,
# chaque individu appartenant à un des 50 groupes
df <- data.frame(id1 = c(1:100),
                 idgpe = sample(50))

# on y applique les instructions de dplyr
df %>% as_tibble() %>%
  mutate(var = rnorm(100)) %>%
  group_by(idgpe) %>%
  summarise(var_mean = mean(var)) -> output_tibble
print(head(output_tibble), 5)
```

Syntaxe dplyr (2)

```
## # A tibble: 5 x 2
##   idgpe var_mean
##   <int>    <dbl>
## 1     1    0.323
## 2     2    0.258
## 3     3    0.179
## 4     4    0.958
## 5     5    0.353
```

Data.table

- Format optimisé de data.frame

Data.table

- Format optimisé de data.frame
- Complémentaire à base R

Data.table

- Format optimisé de data.frame
- Complémentaire à base R
- Optimisation de l'opérateur [

Data.table

- Format optimisé de data.frame
- Complémentaire à base R
- Optimisation de l'opérateur [
- Chaînage possible des instructions

Data.table

- Format optimisé de data.frame
- Complémentaire à base R
- Optimisation de l'opérateur [
- Chaînage possible des instructions
- Plus lisible, plus rapide que base R

Syntaxe data.table (1)

- l'opérateur `[` appliqué au data.table change de signification et devient :

```
DT[i, j, by]
```

Syntaxe data.table (1)

- l'opérateur `[` appliqué au data.table change de signification et devient :

```
DT[i, j, by]
```

- `i` permet de sélectionner des lignes de `DT`

Syntaxe data.table (1)

- l'opérateur `[` appliqué au data.table change de signification et devient :

```
DT[i, j, by]
```

- `i` permet de sélectionner des lignes de `DT`
- `j` permet de créer des variables ou d'en sélectionner

Syntaxe data.table (1)

- l'opérateur `[` appliqué au data.table change de signification et devient :

```
DT[i, j, by]
```

- `i` permet de sélectionner des lignes de `DT`
- `j` permet de créer des variables ou d'en sélectionner
- `by` permet de regrouper les traitement selon les modalités d'une variable définie

Syntaxe data.table (1)

- l'opérateur `[` appliqué au data.table change de signification et devient :

```
DT[i, j, by]
```

- `i` permet de sélectionner des lignes de `DT`
- `j` permet de créer des variables ou d'en sélectionner
- `by` permet de regrouper les traitement selon les modalités d'une variable définie
- L'usage de `[` permet de chaîner les opérations :

Syntaxe data.table (2)

```
library(data.table)
# on convertit notre data frame
# précédemment créé en data.table
dt <- as.data.table(df)

# on y applique les même instructions
dt[, var := rnorm(100)
   ][, list(var_mean = mean(var)),
   by = idgpe] -> output_dt

print(head(output_dt, 5))
```

Syntaxe data.table (2)

```
##     idgpe    var_mean
## 1:     18  0.3872452
## 2:     39  0.2529432
## 3:     34  1.0920601
## 4:     42  0.7632484
## 5:      1 -0.5312545
```

Comparaisons avec base R

`dplyr` et `data.table` présentent un certain nombre d'avantages par rapport à l'usage de base R exclusivement :

- + lisibles et - verbeux, grâce notamment au chaînage
- Pensés pour l'analyse de données
- Instruction optimisées et bien plus rapides que base R

Comparaison des vitesses d'exécution

Étude de cas avec `nycflights13`

- Base `flights` : heures de départ et d'arrivée selon les aéroports + retards au départ et à l'arrivée

Étude de cas avec `nycflights13`

- Base `flights` : heures de départ et d'arrivée selon les aéroports + retards au départ et à l'arrivée
- 336776 lignes et 19 variables

Étude de cas avec `nycflights13`

- Base `flights` : heures de départ et d'arrivée selon les aéroports + retards au départ et à l'arrivée
- 336776 lignes et 19 variables
- Base `weather` : indications météo, heure par heure, dans chaque aéroport

Étude de cas avec `nycflights13`

- Base `flights` : heures de départ et d'arrivée selon les aéroports + retards au départ et à l'arrivée
- 336776 lignes et 19 variables
- Base `weather` : indications météo, heure par heure, dans chaque aéroport
- 26115 lignes et 15 variables

Étude de cas avec `nycflights13`

- Base `flights` : heures de départ et d'arrivée selon les aéroports + retards au départ et à l'arrivée
- 336776 lignes et 19 variables
- Base `weather` : indications météo, heure par heure, dans chaque aéroport
- 26115 lignes et 15 variables
- On crée `flights_dt` et `weather_dt` avec `as.data.table`

Étude de cas avec `nycflights13`

- Base `flights` : heures de départ et d'arrivée selon les aéroports + retards au départ et à l'arrivée
- 336776 lignes et 19 variables
- Base `weather` : indications météo, heure par heure, dans chaque aéroport
- 26115 lignes et 15 variables
- On crée `flights_dt` et `weather_dt` avec `as.data.table`
- Étude de cas : fusion des deux tables pour expliquer retards à l'arrivée et au départ en fonction de la météo

Étude de cas avec nycflights13 - Base R

```
flights_time_hour <- aggregate.data.frame(
  list(arr_delay = flights$arr_delay, dep_delay = flights$dep_
  list(time_hour = flights$time_hour, origin = flights$origin)
  mean)
merge_base <- merge(weather, flights_time_hour, by = c("time_h
```

Étude de cas avec nycflights13 - dplyr

```
flights %>% group_by(time_hour, origin) %>%
  summarise(arr_delay = mean(arr_delay),
            dep_delay = mean(dep_delay)) %>%
  inner_join(weather, by = c("time_hour", "origin")) -> merge_
```

Étude de cas avec nycflights13 - dplyr

```
merge_DT <- merge(
  flights_dt[, list(arr_delay = mean(arr_delay),
                    dep_delay = mean(dep_delay)),
             by = list(time_hour, origin)],
  weather_dt,
  by = c("time_hour", "origin"))
```

Comparaisons des vitesses de ces instructions

Le package `microbenchmark` nous permet de comparer la vitesse de ces instructions :

```
## Unit: milliseconds
##         expr       min         lq      mean    median
##      base R 656.65196 665.40146 700.59051 682.69254 739.914
##       dplyr  51.11633  51.77567  55.16807  53.61034  56.981
##  data.table  24.84791  25.59884  28.43759  26.40666  27.830
##  neval
##     10
##     10
##     10
```